Amendments to the Claims

This listing of claims will replace all prior versions, and listings, of the claims:

1. (Currently amended) A computer system, comprising:

a processor capable of executing multiple threads; and

a main system memory coupled to said processor;

wherein said processor processes a program in a main thread that includes instructions which cause the processor to spawn a pre-execution thread in which only a portion, but not all, of the same program executes, said pre-execution thread runs concurrently with the main thread, but ahead of the main thread in program order, wherein cache misses encountered by the pre-execution thread are resolved before the main thread encounters the cache misses ~~to cause data to be cached before the main thread needs the data~~.

2. (Original) The computer system of claim 1 wherein said instructions that cause the pre-execution thread to be spawned include a start instruction which causes a pre-execution thread to start and a stop instruction which causes the pre-execution thread to stop.

3. (Original) The computer system of claim 2 wherein said start instruction includes a value designating the location in the program where the pre-execution thread is to start running.

4. (Original) The computer system of claim 1 wherein the pre-execution thread encounters a cache miss condition for a memory reference but the main thread does not encounter a cache miss condition when that same memory reference is processed by the main thread.

5. (Original) The computer system of claim 1 wherein said processor determines whether sufficient hardware resources are available before spawning said pre-execution thread.

6. (Original) The computer system of claim 1 wherein said processor ignores exception conditions generated during the pre-execution thread.

7. (Original) The computer system of claim 1 wherein said processor does not permit a store instruction in the pre-execution thread to modify main system memory contents.

8. (Previously presented) The computer system of claim 1 wherein said instructions that cause the pre-execution thread to be spawned include a value that informs a program counter where to begin executing the pre-execution thread.

9. (Original) The computer system of claim 1 further including a buffer into which pre-execution thread stores data is written to make such store data available to pre-execution thread load instructions.

10. (Currently amended) A processor, comprising:
a fetch unit capable of fetching instructions from a plurality of threads;
a program counter coupled to said fetch unit;
an instruction cache coupled to said fetch unit;
a data cache coupled to said instruction cache;
wherein said processor processes a program in a first thread that includes instructions which cause the processor to spawn a second thread in which only a portion, but not all, of the same program also executes, said second thread runs concurrently with the first thread, but ahead of the first thread in program order, wherein ~~so~~ cache misses encountered by the second thread are resolved before the first thread encounters the cache misses.

11. (Original) The processor of claim 10 wherein said instructions that cause a second thread to be spawned include a start instruction which causes the second thread to start and a stop instruction which causes the second thread to stop.

12. (Original) The processor of claim 11 wherein said start instruction includes a value designating the location in the program where the second thread is to start running.

13. (Original) The processor of claim 10 wherein the second thread encounters a cache miss condition for a memory reference but the first thread does not encounter a cache miss condition when that same memory reference is processed by the first thread.

14. (Original) The processor of claim 10 wherein said processor determines whether sufficient hardware resources are available before spawning said second thread.

15. (Original) The processor of claim 10 wherein said processor ignores exception conditions generated during the second thread.

16. (Original) The processor of claim 10 wherein said processor does not permit a store instruction in the second thread to modify data cache contents.

17. (Previously presented) The processor of claim 10 wherein said first and second threads execute same instructions but the second thread executes the same instructions before the first thread to resolve cache misses before the first thread requests data previously subject to a cache miss.

18. (Original) The processor of claim 10 further including a buffer into which pre-execution thread store data is written to make such store data available to pre-execution thread load instructions.

19. (Currently amended) A method of running a program in a processor, comprising:
(a) inserting pre-execution thread instructions in the program;

(b) spawning a pre-execution thread when designated by the inserted instructions; and

(c) running said pre-execution thread concurrently with a main thread wherein both the pre-execution and the main threads include instructions from the same program, the pre-execution thread running ahead of the main thread and containing only a portion of the instructions from the main thread, wherein ~~so~~ cache misses encountered by the pre-execution thread are resolved before the main thread encounters the cache misses.

20. (Original) The method of claim 19 further including stopping the pre-execution thread when designated by the inserted instructions in (a).

21. (Original) The method of claim 19 further including copying register contents associated with the main thread to registers used by the pre-execution thread.

22. (Original) The method of claim 19 further including determining whether sufficient hardware resources are available to spawn the pre-execution thread.

23. (Original) The method of claim 19 further including ignoring all exception conditions generated during the pre-execution thread.

24. (Previously presented) The method of claim 19 further including creating sufficient slack in time for executing same instructions between the main thread and the pre-execution thread so the pre-execution thread has time to resolve cache misses before the main thread requests data previously subject to a cache miss.

25. (Original) The method of claim 19 further including copying the contents of at least one register to memory to make such contents available to the pre-execution thread.

26. (Original) The method of claim 19 further including writing pre-execution store data into a buffer that can be accessed by a pre-execution load instruction.

27. (Original) The method of claim 19 further including not permitting any store instruction in the pre-execution thread to modify memory contents.

28. (Original) The method of claim 19 wherein (a) includes inserting a start instruction which causes the processor to start the pre-execution thread.

29. (Original) The method of claim 19 wherein the start instruction specifies a location in the program at which the pre-execution thread is to start running.

30. (Original) The method of claim 19 wherein (a) includes inserting a stop instruction which causes the processor stop the pre-execution thread.

31. (Currently amended) A processor, comprising:

a fetch unit capable of fetching instructions from a plurality of threads;

a program counter coupled to said fetch unit;

an instruction cache coupled to said fetch unit;

a data cache coupled to said instruction cache;

wherein, in a pre-execution thread, said processor pre-executes instructions from a main thread that are specified by said main thread, wherein ~~so~~ a cache miss encountered by the pre-execution thread is resolved before the main thread encounters the cache miss.

32. (Previously presented) The processor of claim 31 wherein the pre-execution thread is caused to be spawned to pre-execute said instructions by an instruction in the main thread.

33. (Previously presented) The processor of claim 31 wherein the pre-execution thread spins on a variable that is set to a predetermined value by the main thread when there are instructions to pre-execute.

34. (Original) The processor of claim 31 wherein the processor ceases pre-executing instructions when a program counter is encountered that exceeds a range.

35. (Original) The processor of claim 31 wherein the processor ceases pre-executing instructions when the main thread catches up to the pre-executing instructions.

36. (Original) The processor of claim 31 wherein the processor ceases pre-executing instructions when the number of pre-executing instructions exceeds a limit.